Practice Lab: ResNet architecture for Intel Image Classification

We will use a ResNet architecture to solve an imagery classification problem. You have to work with the Intel Image Classification, which consists of 6 classes with 25k samples.

Outline

1 - Packages

In [10]:
```
import numpy as np
import pandas as pd
import os
import shutil
import matplotlib.pyplot as plt
from PIL import Image
from PIL import ImageFilter
from sklearn.preprocessing import LabelEncoder
import timeit
import cv2 as cv2
import argparse
from sklearn.utils import shuffle

#Neural Network packages

import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from torch.utils.data.sampler import SubsetRandomSampler
from torch.utils.data import Dataset
from torchvision import models
from torch.autograd import Variable
```

In [ ]:
```
```

Using kaggle this time

In [2]:
```
os.getcwd()
```

Out[2]:
```
'/kaggle/working'
```

GPU P100

In [3]:
```
device= torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(device, torch.cuda.get_device_name(0))
```

```
cuda Tesla P100-PCIE-16GB
```

2 - Loading Data and Transform it

In [49]:
```
transform0 = transforms.Compose([transforms.Resize(256),
                                transforms.RandomHorizontalFlip(),
                                transforms.CenterCrop(224),
                                transforms.ToTensor(),
                                transforms.Normalize([0,0,0],
                                                     [1,1,1])])
```

In [54]:
```
transform1 = transforms.Compose([transforms.Resize(256),
                                transforms.RandomHorizontalFlip(),
                                transforms.RandomVerticalFlip(),
                                transforms.RandomRotation(degrees=[0,90]),
                                transforms.CenterCrop(224),
                                #transforms.TenCrop(224),
                                transforms.ToTensor(),
                                transforms.Normalize([0,0,0],
                                                     [1,1,1])])
```

In [5]:
```
# Custom Dataset Class
class INTELDataset(Dataset):
    def __init__(self, img_data,img_path,transform=None):
        self.img_path = img_path
        self.transform = transform
        self.img_data = img_data



    def __getitem__(self, index):
        img_name = os.path.join(self.img_path,self.img_data.loc[index, 'labels'],
                                self.img_data.loc[index, 'Images'])
        image = Image.open(img_name)
        image = image.convert('RGB')
        #image = image.resize((300,300))
        label = torch.tensor(self.img_data.loc[index, 'encoded_labels'])
        if self.transform is not None:
            image = self.transform(image)
        return image, label
```

```
    def __len__(self):
        return len(self.img_data)
```

Plotting Data

In [8]:

```
def get_images(directory):
    Images = []
    Labels = []
    label = 0
    for labels in os.listdir(".//Dataset/seg_train/seg_train"): #Main Directory where e
        if labels == 'glacier': #Folder contain Glacier Images get the '2' class label.
            label = 2
        elif labels == 'sea':
            label = 4
        elif labels == 'buildings':
            label = 0
        elif labels == 'forest':

            label = 1
        elif labels == 'street':
            label = 5
        elif labels == 'mountain':
            label = 3

        for image_file in os.listdir(".//Dataset/seg_train/seg_train/"+labels): #Extrac
            image = cv2.imread(".//Dataset/seg_train/seg_train/"+labels+r'/'+image_file
            image = cv2.resize(image,(150,150)) #Resize the image, Some images are diff
            Images.append(image)
            Labels.append(label)

    return shuffle(Images,Labels,random_state=817328462) #Shuffle the dataset you just

def get_classlabel(class_code):
    labels = {2:'glacier', 4:'sea', 0:'buildings', 1:'forest', 5:'street', 3:'mountain'

    return labels[class_code]
```

In [11]:

```
Images, Labels = get_images('../input/seg_train/seg_train/') #Extract the training imag

TrainImages = np.array(Images) #converting the list of images to numpy array.
TrainLabels = np.array(Labels)
```

In [12]:

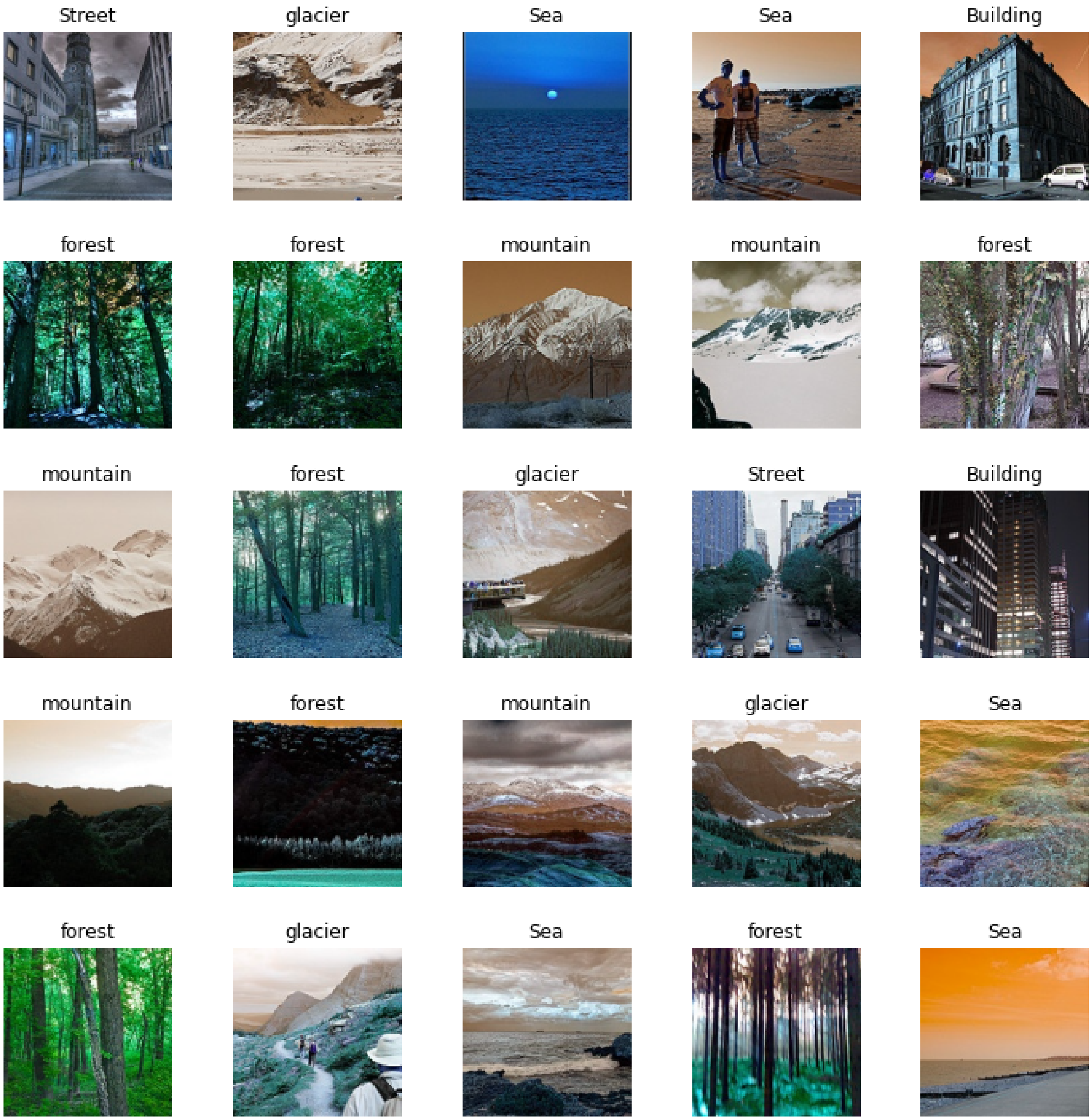
```
Labels = [ "Building",
          "forest",
          "glacier",
          "mountain",
          "Sea",
          "Street"
]
```

In [13]:

```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
X = TrainImages
```

```
fig, axes = plt.subplots(5,5, figsize=(10,10))
fig.tight_layout(pad=0.1)

for i,ax in enumerate(axes.flat):
    random_index = np.random.randint(14034)
    ax.imshow(X[random_index], cmap='gray')
    ax.set_title(Labels[int(TrainLabels[random_index])])
    ax.set_axis_off()
```



Creating training data labels

In [6]:
```
trainingPath = '../input/intel-image-classification/seg_train'
testingPath  = '../input/intel-image-classification/seg_test'
```

seperate labels and images then convert it to a dataframe !

In [8]:

```
images_train=[]
labels_train=[]


for file in os.listdir(os.path.join(trainingPath,'seg_train')):
  if file=='buildings':
    for img in os.listdir(os.path.join(trainingPath,'seg_train', file)):
      images_train.append(img)
      labels_train.append('buildings')
  if file=='forest':
    for img in os.listdir(os.path.join(trainingPath,'seg_train', file)):
      images_train.append(img)
      labels_train.append('forest')
  if file=='glacier':
    for img in os.listdir(os.path.join(trainingPath,'seg_train', file)):
      images_train.append(img)
      labels_train.append('glacier')
  if file=='mountain':
    for img in os.listdir(os.path.join(trainingPath,'seg_train', file)):
      images_train.append(img)
      labels_train.append('mountain')
  if file=='sea':
    for img in os.listdir(os.path.join(trainingPath,'seg_train', file)):
      images_train.append(img)
      labels_train.append('sea')
  if file=='street':
    for img in os.listdir(os.path.join(trainingPath,'seg_train', file)):
      images_train.append(img)
      labels_train.append('street')

data_train= {'Images': images_train, 'labels':labels_train}
data_train=pd.DataFrame(data_train)
print(data_train.head(15))
```

```
       Images    labels
0   14986.jpg  mountain
1    3138.jpg  mountain
2    1700.jpg  mountain
3   16257.jpg  mountain
4    2863.jpg  mountain
5     771.jpg  mountain
6   12167.jpg  mountain
7   17643.jpg  mountain
8    6560.jpg  mountain
9   10162.jpg  mountain
10   4009.jpg  mountain
11  15823.jpg  mountain
12    820.jpg  mountain
13   6272.jpg  mountain
14  15783.jpg  mountain
```

Convert string labels to int

In [9]:

```
encoder= LabelEncoder()
data_train['encoded_labels']= encoder.fit_transform(data_train['labels'])
data_train.head(15)
data_train.encoded_labels.value_counts()
```

Out[9]:
```
3    2512
2    2404
5    2382
4    2274
1    2271
0    2191
Name: encoded_labels, dtype: int64
```

In [11]:
```
train_batch_size =32
random_seed      =50
```

In [12]:
```
# Create train sampler
train_indices = list(range(len(data_train)))

train_sampler = SubsetRandomSampler(train_indices)
```

In [13]:
```
#training image path
path_train=os.path.join(trainingPath,'seg_train')
print(path_train)
```

```
../input/intel-image-classification/seg_train/seg_train
```

In [55]:
```
dataset_train = INTELDataset(data_train,path_train,transform1)



train_loader = torch.utils.data.DataLoader(dataset_train, batch_size=train_batch_size,
                                           sampler=train_sampler)
```

Now for the testingSet

In [15]:
```
images_test=[]
labels_test=[]


for file in os.listdir(os.path.join(testingPath,'seg_test')):
  if file=='buildings':
    for img in os.listdir(os.path.join(testingPath,'seg_test', file)):
      images_test.append(img)
      labels_test.append('buildings')
  if file=='forest':
    for img in os.listdir(os.path.join(testingPath,'seg_test', file)):
      images_test.append(img)
      labels_test.append('forest')
  if file=='glacier':
    for img in os.listdir(os.path.join(testingPath,'seg_test', file)):
      images_test.append(img)
      labels_test.append('glacier')
  if file=='mountain':
    for img in os.listdir(os.path.join(testingPath,'seg_test', file)):
      images_test.append(img)
      labels_test.append('mountain')
  if file=='sea':
    for img in os.listdir(os.path.join(testingPath,'seg_test', file)):
```

```
            images_test.append(img)
            labels_test.append('sea')
    if file=='street':
        for img in os.listdir(os.path.join(testingPath,'seg_test', file)):
            images_test.append(img)
            labels_test.append('street')

data_test= {'Images': images_test, 'labels':labels_test}
data_test=pd.DataFrame(data_test)
print(data_test.head(10))
```

```
      Images    labels
0  22608.jpg  mountain
1  23274.jpg  mountain
2  23775.jpg  mountain
3  22046.jpg  mountain
4  23436.jpg  mountain
5  20684.jpg  mountain
6  20554.jpg  mountain
7  21093.jpg  mountain
8  24287.jpg  mountain
9  20762.jpg  mountain
```

In [16]:

```
encoder= LabelEncoder()
data_test['encoded_labels']= encoder.fit_transform(data_test['labels'])
data_test.head(10)
data_test.encoded_labels.value_counts()
```

Out[16]:

```
2    553
3    525
4    510
5    501
1    474
0    437
Name: encoded_labels, dtype: int64
```

In [17]:

```
path_test=os.path.join(testingPath,'seg_test')
```

In [18]:

```
# Creating Test Sampler
test_indices = list(range(len(data_test)))

test_sampler = SubsetRandomSampler(test_indices)
```

In [19]:

```
test_batch_size=20
```

In [56]:

```
dataset_test = INTELDataset(data_test,path_test,transform1)



test_loader = torch.utils.data.DataLoader(dataset_test, batch_size=test_batch_size,
                                           sampler=test_sampler)
```

Function for Training

In [21]:

```
def train(model, device, train_loader, criterion, optimizer, epoch):
    model.cuda()
    model.train()
    train_losses=[]
    for batch_idx, (data, target) in enumerate(train_loader):

        # send the image, target to the device
        data, target = data.to(device), target.to(device)
        # flush out the gradients stored in optimizer
        optimizer.zero_grad()
        # pass the image to the model and assign the output to variable named output
        output = model(data)
        # calculate the loss (use cross entropy in pytorch)
        loss = criterion(output, target)
        #appending train loss list
        train_losses.append(loss.item())
        # do a backward pass
        loss.backward()
        # update the weights
        optimizer.step()

        if batch_idx % 32 == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))

    return train_losses
```

Function for Testing

In [22]:

```
def test(model, device, test_loader, criterion):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:

            # send the image, target to the device
            data, target = data.to(device), target.to(device)
            # pass the image to the model and assign the output to variable named outpu
            output = model(data)
            #print(output)

            test_loss += criterion(output, target).item()  # sum up batch loss
            pred = output.argmax(dim=1, keepdim=True)  # get the index of the max log-p
            correct += pred.eq(target.view_as(pred)).sum().item()
    #print(output)
    test_loss /= len(test_loader.dataset)

    print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
        test_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)))
```

Training and Testing using pretrained Resnet50

In [ ]:

```
model = models.resnet50(pretrained=True)
for param in model.parameters():
    param.requires_grad = False
num_features= model.fc.in_features
model.fc = nn.Linear(num_features, 6)


criterion = nn.CrossEntropyLoss().cuda()
#Define Adam Optimiser with a learning rate of 0.01
optimizer = optim.Adam(model.fc.parameters(), lr=0.01)

start = timeit.default_timer()
for epoch in range(1, 11):
    train(model, device, train_loader, criterion, optimizer, epoch)
    test(model, device, test_loader, criterion)
stop = timeit.default_timer()
print('Total time taken: {} seconds'.format(int(stop - start)) )
```

In [46]:

```
model = models.resnet50(pretrained=True)
for param in model.parameters():
    param.requires_grad = False
num_features= model.fc.in_features
model.fc = nn.Linear(num_features, 6)


criterion = nn.CrossEntropyLoss().cuda()
#Define Adam Optimiser with a learning rate of 0.01
optimizer = optim.Adam(model.fc.parameters(), lr=0.01)

start = timeit.default_timer()
for epoch in range(1, 11):
    train(model, device, train_loader, criterion, optimizer, epoch)
    test(model, device, test_loader, criterion)
stop = timeit.default_timer()
print('Total time taken: {} seconds'.format(int(stop - start)) )
```

```
Train Epoch: 1 [0/14034 (0%)]	Loss: 1.830000
Train Epoch: 1 [1024/14034 (7%)]	Loss: 0.768932
Train Epoch: 1 [2048/14034 (15%)]	Loss: 1.148619
Train Epoch: 1 [3072/14034 (22%)]	Loss: 0.556057
Train Epoch: 1 [4096/14034 (29%)]	Loss: 0.502832
Train Epoch: 1 [5120/14034 (36%)]	Loss: 0.633436
Train Epoch: 1 [6144/14034 (44%)]	Loss: 0.541143
Train Epoch: 1 [7168/14034 (51%)]	Loss: 1.178295
Train Epoch: 1 [8192/14034 (58%)]	Loss: 0.705637
Train Epoch: 1 [9216/14034 (66%)]	Loss: 0.585259
Train Epoch: 1 [10240/14034 (73%)]	Loss: 0.846293
Train Epoch: 1 [11264/14034 (80%)]	Loss: 1.050159
Train Epoch: 1 [12288/14034 (87%)]	Loss: 1.283498
Train Epoch: 1 [13312/14034 (95%)]	Loss: 0.517139

Test set: Average loss: 0.0496, Accuracy: 2202/3000 (73%)
```

```
Train Epoch: 2 [0/14034 (0%)]	Loss: 1.644198
Train Epoch: 2 [1024/14034 (7%)]	Loss: 0.485926
Train Epoch: 2 [2048/14034 (15%)]	Loss: 0.941686
Train Epoch: 2 [3072/14034 (22%)]	Loss: 0.935783
Train Epoch: 2 [4096/14034 (29%)]	Loss: 1.350318
Train Epoch: 2 [5120/14034 (36%)]	Loss: 1.070438
Train Epoch: 2 [6144/14034 (44%)]	Loss: 0.539926
Train Epoch: 2 [7168/14034 (51%)]	Loss: 0.870158
Train Epoch: 2 [8192/14034 (58%)]	Loss: 0.797639
Train Epoch: 2 [9216/14034 (66%)]	Loss: 0.896651
Train Epoch: 2 [10240/14034 (73%)]	Loss: 1.590058
Train Epoch: 2 [11264/14034 (80%)]	Loss: 0.860574
Train Epoch: 2 [12288/14034 (87%)]	Loss: 0.889665
Train Epoch: 2 [13312/14034 (95%)]	Loss: 0.356661

Test set: Average loss: 0.0291, Accuracy: 2513/3000 (84%)

Train Epoch: 3 [0/14034 (0%)]	Loss: 0.745318
Train Epoch: 3 [1024/14034 (7%)]	Loss: 0.771138
Train Epoch: 3 [2048/14034 (15%)]	Loss: 0.681193
Train Epoch: 3 [3072/14034 (22%)]	Loss: 0.953363
Train Epoch: 3 [4096/14034 (29%)]	Loss: 0.921867
Train Epoch: 3 [5120/14034 (36%)]	Loss: 0.778117
Train Epoch: 3 [6144/14034 (44%)]	Loss: 1.111316
Train Epoch: 3 [7168/14034 (51%)]	Loss: 0.946807
Train Epoch: 3 [8192/14034 (58%)]	Loss: 1.331000
Train Epoch: 3 [9216/14034 (66%)]	Loss: 0.244928
Train Epoch: 3 [10240/14034 (73%)]	Loss: 0.467147
Train Epoch: 3 [11264/14034 (80%)]	Loss: 0.735551
Train Epoch: 3 [12288/14034 (87%)]	Loss: 1.097497
Train Epoch: 3 [13312/14034 (95%)]	Loss: 0.961249

Test set: Average loss: 0.0314, Accuracy: 2525/3000 (84%)

Train Epoch: 4 [0/14034 (0%)]	Loss: 0.709621
Train Epoch: 4 [1024/14034 (7%)]	Loss: 1.527600
Train Epoch: 4 [2048/14034 (15%)]	Loss: 1.358303
Train Epoch: 4 [3072/14034 (22%)]	Loss: 1.490046
Train Epoch: 4 [4096/14034 (29%)]	Loss: 0.673652
Train Epoch: 4 [5120/14034 (36%)]	Loss: 0.826581
Train Epoch: 4 [6144/14034 (44%)]	Loss: 0.615865
Train Epoch: 4 [7168/14034 (51%)]	Loss: 0.438259
Train Epoch: 4 [8192/14034 (58%)]	Loss: 1.190838
Train Epoch: 4 [9216/14034 (66%)]	Loss: 2.582569
Train Epoch: 4 [10240/14034 (73%)]	Loss: 0.775833
Train Epoch: 4 [11264/14034 (80%)]	Loss: 0.631050
Train Epoch: 4 [12288/14034 (87%)]	Loss: 0.760836
Train Epoch: 4 [13312/14034 (95%)]	Loss: 0.860425

Test set: Average loss: 0.0271, Accuracy: 2622/3000 (87%)

Train Epoch: 5 [0/14034 (0%)]	Loss: 0.872145
Train Epoch: 5 [1024/14034 (7%)]	Loss: 0.924736
Train Epoch: 5 [2048/14034 (15%)]	Loss: 1.343746
Train Epoch: 5 [3072/14034 (22%)]	Loss: 1.181394
Train Epoch: 5 [4096/14034 (29%)]	Loss: 1.975080
Train Epoch: 5 [5120/14034 (36%)]	Loss: 0.983690
Train Epoch: 5 [6144/14034 (44%)]	Loss: 0.109590
Train Epoch: 5 [7168/14034 (51%)]	Loss: 0.641294
```

```
Train Epoch: 5 [8192/14034 (58%)]	Loss: 1.274766
Train Epoch: 5 [9216/14034 (66%)]	Loss: 0.506959
Train Epoch: 5 [10240/14034 (73%)]	Loss: 0.425260
Train Epoch: 5 [11264/14034 (80%)]	Loss: 0.546069
Train Epoch: 5 [12288/14034 (87%)]	Loss: 0.753780
Train Epoch: 5 [13312/14034 (95%)]	Loss: 1.151840

Test set: Average loss: 0.0428, Accuracy: 2452/3000 (82%)

Train Epoch: 6 [0/14034 (0%)]	Loss: 0.734784
Train Epoch: 6 [1024/14034 (7%)]	Loss: 0.940118
Train Epoch: 6 [2048/14034 (15%)]	Loss: 3.254449
Train Epoch: 6 [3072/14034 (22%)]	Loss: 1.111667
Train Epoch: 6 [4096/14034 (29%)]	Loss: 0.778687
Train Epoch: 6 [5120/14034 (36%)]	Loss: 0.304910
Train Epoch: 6 [6144/14034 (44%)]	Loss: 1.692008
Train Epoch: 6 [7168/14034 (51%)]	Loss: 1.707589
Train Epoch: 6 [8192/14034 (58%)]	Loss: 0.878980
Train Epoch: 6 [9216/14034 (66%)]	Loss: 0.723250
Train Epoch: 6 [10240/14034 (73%)]	Loss: 2.642448
Train Epoch: 6 [11264/14034 (80%)]	Loss: 0.036429
Train Epoch: 6 [12288/14034 (87%)]	Loss: 0.404901
Train Epoch: 6 [13312/14034 (95%)]	Loss: 1.250604

Test set: Average loss: 0.0561, Accuracy: 2388/3000 (80%)

Train Epoch: 7 [0/14034 (0%)]	Loss: 1.772076
Train Epoch: 7 [1024/14034 (7%)]	Loss: 2.003261
Train Epoch: 7 [2048/14034 (15%)]	Loss: 1.679632
Train Epoch: 7 [3072/14034 (22%)]	Loss: 0.488299
Train Epoch: 7 [4096/14034 (29%)]	Loss: 1.106084
Train Epoch: 7 [5120/14034 (36%)]	Loss: 2.039451
Train Epoch: 7 [6144/14034 (44%)]	Loss: 0.961563
Train Epoch: 7 [7168/14034 (51%)]	Loss: 0.195244
Train Epoch: 7 [8192/14034 (58%)]	Loss: 2.101213
Train Epoch: 7 [9216/14034 (66%)]	Loss: 0.538220
Train Epoch: 7 [10240/14034 (73%)]	Loss: 1.447686
Train Epoch: 7 [11264/14034 (80%)]	Loss: 0.337975
Train Epoch: 7 [12288/14034 (87%)]	Loss: 1.308509
Train Epoch: 7 [13312/14034 (95%)]	Loss: 1.467153

Test set: Average loss: 0.0419, Accuracy: 2542/3000 (85%)

Train Epoch: 8 [0/14034 (0%)]	Loss: 1.815910
Train Epoch: 8 [1024/14034 (7%)]	Loss: 0.575189
Train Epoch: 8 [2048/14034 (15%)]	Loss: 0.918404
Train Epoch: 8 [3072/14034 (22%)]	Loss: 2.593571
Train Epoch: 8 [4096/14034 (29%)]	Loss: 1.238541
Train Epoch: 8 [5120/14034 (36%)]	Loss: 0.673494
Train Epoch: 8 [6144/14034 (44%)]	Loss: 1.775759
Train Epoch: 8 [7168/14034 (51%)]	Loss: 0.269467
Train Epoch: 8 [8192/14034 (58%)]	Loss: 1.053019
Train Epoch: 8 [9216/14034 (66%)]	Loss: 1.545264
Train Epoch: 8 [10240/14034 (73%)]	Loss: 1.068793
Train Epoch: 8 [11264/14034 (80%)]	Loss: 1.187988
Train Epoch: 8 [12288/14034 (87%)]	Loss: 1.452800
Train Epoch: 8 [13312/14034 (95%)]	Loss: 0.086830

Test set: Average loss: 0.0352, Accuracy: 2558/3000 (85%)
```

```
Train Epoch: 9 [0/14034 (0%)]	Loss: 1.171262
Train Epoch: 9 [1024/14034 (7%)]	Loss: 0.381122
Train Epoch: 9 [2048/14034 (15%)]	Loss: 0.369459
Train Epoch: 9 [3072/14034 (22%)]	Loss: 0.735076
Train Epoch: 9 [4096/14034 (29%)]	Loss: 0.539967
Train Epoch: 9 [5120/14034 (36%)]	Loss: 1.222156
Train Epoch: 9 [6144/14034 (44%)]	Loss: 1.015466
Train Epoch: 9 [7168/14034 (51%)]	Loss: 0.924311
Train Epoch: 9 [8192/14034 (58%)]	Loss: 1.162696
Train Epoch: 9 [9216/14034 (66%)]	Loss: 0.658135
Train Epoch: 9 [10240/14034 (73%)]	Loss: 1.415770
Train Epoch: 9 [11264/14034 (80%)]	Loss: 1.066526
Train Epoch: 9 [12288/14034 (87%)]	Loss: 0.149793
Train Epoch: 9 [13312/14034 (95%)]	Loss: 1.205858

Test set: Average loss: 0.0247, Accuracy: 2635/3000 (88%)

Train Epoch: 10 [0/14034 (0%)]	Loss: 1.307477
Train Epoch: 10 [1024/14034 (7%)]	Loss: 1.640177
Train Epoch: 10 [2048/14034 (15%)]	Loss: 1.236343
Train Epoch: 10 [3072/14034 (22%)]	Loss: 1.151335
Train Epoch: 10 [4096/14034 (29%)]	Loss: 0.820674
Train Epoch: 10 [5120/14034 (36%)]	Loss: 1.213505
Train Epoch: 10 [6144/14034 (44%)]	Loss: 1.591729
Train Epoch: 10 [7168/14034 (51%)]	Loss: 1.811929
Train Epoch: 10 [8192/14034 (58%)]	Loss: 1.093415
Train Epoch: 10 [9216/14034 (66%)]	Loss: 0.780130
Train Epoch: 10 [10240/14034 (73%)]	Loss: 0.944442
Train Epoch: 10 [11264/14034 (80%)]	Loss: 0.955792
Train Epoch: 10 [12288/14034 (87%)]	Loss: 0.682969
Train Epoch: 10 [13312/14034 (95%)]	Loss: 1.604892

Test set: Average loss: 0.0363, Accuracy: 2574/3000 (86%)

Total time taken: 936 seconds
```

In [57]:

```python
model = models.resnet50(pretrained=True)
for param in model.parameters():
    param.requires_grad = False
num_features= model.fc.in_features
model.fc = nn.Linear(num_features, 6)


criterion = nn.CrossEntropyLoss().cuda()
#Define Adam Optimiser with a learning rate of 0.01
optimizer = optim.Adam(model.fc.parameters(), lr=0.01)

start = timeit.default_timer()
for epoch in range(1, 11):
    train(model, device, train_loader, criterion, optimizer, epoch)
    test(model, device, test_loader, criterion)
stop = timeit.default_timer()
print('Total time taken: {} seconds'.format(int(stop - start)) )
```

```
Train Epoch: 1 [0/14034 (0%)]	Loss: 1.821805
Train Epoch: 1 [1024/14034 (7%)]	Loss: 0.370200
Train Epoch: 1 [2048/14034 (15%)]	Loss: 1.337283
Train Epoch: 1 [3072/14034 (22%)]	Loss: 0.556116
Train Epoch: 1 [4096/14034 (29%)]	Loss: 0.804838
```

```
Train Epoch: 1 [5120/14034 (36%)]	Loss: 1.477892
Train Epoch: 1 [6144/14034 (44%)]	Loss: 0.474627
Train Epoch: 1 [7168/14034 (51%)]	Loss: 0.655048
Train Epoch: 1 [8192/14034 (58%)]	Loss: 0.985758
Train Epoch: 1 [9216/14034 (66%)]	Loss: 1.121586
Train Epoch: 1 [10240/14034 (73%)]	Loss: 0.555382
Train Epoch: 1 [11264/14034 (80%)]	Loss: 1.009061
Train Epoch: 1 [12288/14034 (87%)]	Loss: 1.363163
Train Epoch: 1 [13312/14034 (95%)]	Loss: 1.116066

Test set: Average loss: 0.0286, Accuracy: 2495/3000 (83%)

Train Epoch: 2 [0/14034 (0%)]	Loss: 0.751358
Train Epoch: 2 [1024/14034 (7%)]	Loss: 1.222518
Train Epoch: 2 [2048/14034 (15%)]	Loss: 2.090687
Train Epoch: 2 [3072/14034 (22%)]	Loss: 0.521674
Train Epoch: 2 [4096/14034 (29%)]	Loss: 1.461105
Train Epoch: 2 [5120/14034 (36%)]	Loss: 0.980768
Train Epoch: 2 [6144/14034 (44%)]	Loss: 0.834417
Train Epoch: 2 [7168/14034 (51%)]	Loss: 1.353793
Train Epoch: 2 [8192/14034 (58%)]	Loss: 1.287991
Train Epoch: 2 [9216/14034 (66%)]	Loss: 0.413238
Train Epoch: 2 [10240/14034 (73%)]	Loss: 1.224951
Train Epoch: 2 [11264/14034 (80%)]	Loss: 2.199096
Train Epoch: 2 [12288/14034 (87%)]	Loss: 1.105188
Train Epoch: 2 [13312/14034 (95%)]	Loss: 0.510176

Test set: Average loss: 0.0374, Accuracy: 2469/3000 (82%)

Train Epoch: 3 [0/14034 (0%)]	Loss: 0.517898
Train Epoch: 3 [1024/14034 (7%)]	Loss: 0.814740
Train Epoch: 3 [2048/14034 (15%)]	Loss: 1.197475
Train Epoch: 3 [3072/14034 (22%)]	Loss: 0.494846
Train Epoch: 3 [4096/14034 (29%)]	Loss: 1.052792
Train Epoch: 3 [5120/14034 (36%)]	Loss: 1.345079
Train Epoch: 3 [6144/14034 (44%)]	Loss: 1.070615
Train Epoch: 3 [7168/14034 (51%)]	Loss: 0.460959
Train Epoch: 3 [8192/14034 (58%)]	Loss: 1.033654
Train Epoch: 3 [9216/14034 (66%)]	Loss: 0.991731
Train Epoch: 3 [10240/14034 (73%)]	Loss: 0.827790
Train Epoch: 3 [11264/14034 (80%)]	Loss: 0.317698
Train Epoch: 3 [12288/14034 (87%)]	Loss: 0.515733
Train Epoch: 3 [13312/14034 (95%)]	Loss: 1.808995

Test set: Average loss: 0.0442, Accuracy: 2380/3000 (79%)

Train Epoch: 4 [0/14034 (0%)]	Loss: 0.596928
Train Epoch: 4 [1024/14034 (7%)]	Loss: 0.765893
Train Epoch: 4 [2048/14034 (15%)]	Loss: 0.926294
Train Epoch: 4 [3072/14034 (22%)]	Loss: 1.698914
Train Epoch: 4 [4096/14034 (29%)]	Loss: 0.549599
Train Epoch: 4 [5120/14034 (36%)]	Loss: 0.745216
Train Epoch: 4 [6144/14034 (44%)]	Loss: 2.122044
Train Epoch: 4 [7168/14034 (51%)]	Loss: 0.567969
Train Epoch: 4 [8192/14034 (58%)]	Loss: 1.096119
Train Epoch: 4 [9216/14034 (66%)]	Loss: 0.696394
Train Epoch: 4 [10240/14034 (73%)]	Loss: 0.745209
Train Epoch: 4 [11264/14034 (80%)]	Loss: 1.552247
Train Epoch: 4 [12288/14034 (87%)]	Loss: 0.893436
Train Epoch: 4 [13312/14034 (95%)]	Loss: 1.450641
```

```
Test set: Average loss: 0.0471, Accuracy: 2386/3000 (80%)

Train Epoch: 5 [0/14034 (0%)]   Loss: 0.827842
Train Epoch: 5 [1024/14034 (7%)]        Loss: 1.319914
Train Epoch: 5 [2048/14034 (15%)]       Loss: 0.418737
Train Epoch: 5 [3072/14034 (22%)]       Loss: 0.699504
Train Epoch: 5 [4096/14034 (29%)]       Loss: 0.390827
Train Epoch: 5 [5120/14034 (36%)]       Loss: 0.457933
Train Epoch: 5 [6144/14034 (44%)]       Loss: 1.473841
Train Epoch: 5 [7168/14034 (51%)]       Loss: 0.905394
Train Epoch: 5 [8192/14034 (58%)]       Loss: 0.781033
Train Epoch: 5 [9216/14034 (66%)]       Loss: 1.579053
Train Epoch: 5 [10240/14034 (73%)]      Loss: 1.975147
Train Epoch: 5 [11264/14034 (80%)]      Loss: 1.210608
Train Epoch: 5 [12288/14034 (87%)]      Loss: 0.334912
Train Epoch: 5 [13312/14034 (95%)]      Loss: 0.754901

Test set: Average loss: 0.0301, Accuracy: 2550/3000 (85%)

Train Epoch: 6 [0/14034 (0%)]   Loss: 0.578655
Train Epoch: 6 [1024/14034 (7%)]        Loss: 1.242607
Train Epoch: 6 [2048/14034 (15%)]       Loss: 0.683522
Train Epoch: 6 [3072/14034 (22%)]       Loss: 0.467343
Train Epoch: 6 [4096/14034 (29%)]       Loss: 0.873508
Train Epoch: 6 [5120/14034 (36%)]       Loss: 0.909274
Train Epoch: 6 [6144/14034 (44%)]       Loss: 1.073887
Train Epoch: 6 [7168/14034 (51%)]       Loss: 1.053207
Train Epoch: 6 [8192/14034 (58%)]       Loss: 1.208908
Train Epoch: 6 [9216/14034 (66%)]       Loss: 1.922277
Train Epoch: 6 [10240/14034 (73%)]      Loss: 0.317112
Train Epoch: 6 [11264/14034 (80%)]      Loss: 0.393317
Train Epoch: 6 [12288/14034 (87%)]      Loss: 0.985411
Train Epoch: 6 [13312/14034 (95%)]      Loss: 1.716823

Test set: Average loss: 0.0427, Accuracy: 2492/3000 (83%)

Train Epoch: 7 [0/14034 (0%)]   Loss: 1.510128
Train Epoch: 7 [1024/14034 (7%)]        Loss: 0.837584
Train Epoch: 7 [2048/14034 (15%)]       Loss: 0.428996
Train Epoch: 7 [3072/14034 (22%)]       Loss: 1.497144
Train Epoch: 7 [4096/14034 (29%)]       Loss: 0.893721
Train Epoch: 7 [5120/14034 (36%)]       Loss: 1.251913
Train Epoch: 7 [6144/14034 (44%)]       Loss: 1.162032
Train Epoch: 7 [7168/14034 (51%)]       Loss: 0.799297
Train Epoch: 7 [8192/14034 (58%)]       Loss: 0.517682
Train Epoch: 7 [9216/14034 (66%)]       Loss: 1.182894
Train Epoch: 7 [10240/14034 (73%)]      Loss: 0.945793
Train Epoch: 7 [11264/14034 (80%)]      Loss: 1.043706
Train Epoch: 7 [12288/14034 (87%)]      Loss: 1.655334
Train Epoch: 7 [13312/14034 (95%)]      Loss: 0.314348

Test set: Average loss: 0.0795, Accuracy: 2205/3000 (74%)

Train Epoch: 8 [0/14034 (0%)]   Loss: 2.591330
Train Epoch: 8 [1024/14034 (7%)]        Loss: 0.730492
Train Epoch: 8 [2048/14034 (15%)]       Loss: 0.981598
Train Epoch: 8 [3072/14034 (22%)]       Loss: 0.799946
Train Epoch: 8 [4096/14034 (29%)]       Loss: 1.122170
Train Epoch: 8 [5120/14034 (36%)]       Loss: 0.998156
```

```
Train Epoch: 8 [6144/14034 (44%)]       Loss: 1.619675
Train Epoch: 8 [7168/14034 (51%)]       Loss: 1.100064
Train Epoch: 8 [8192/14034 (58%)]       Loss: 0.608468
Train Epoch: 8 [9216/14034 (66%)]       Loss: 0.753398
Train Epoch: 8 [10240/14034 (73%)]      Loss: 2.547336
Train Epoch: 8 [11264/14034 (80%)]      Loss: 1.079462
Train Epoch: 8 [12288/14034 (87%)]      Loss: 1.241430
Train Epoch: 8 [13312/14034 (95%)]      Loss: 0.133334

Test set: Average loss: 0.0346, Accuracy: 2553/3000 (85%)

Train Epoch: 9 [0/14034 (0%)]   Loss: 1.447429
Train Epoch: 9 [1024/14034 (7%)]        Loss: 1.204093
Train Epoch: 9 [2048/14034 (15%)]       Loss: 1.954382
Train Epoch: 9 [3072/14034 (22%)]       Loss: 1.312523
Train Epoch: 9 [4096/14034 (29%)]       Loss: 1.019283
Train Epoch: 9 [5120/14034 (36%)]       Loss: 0.694343
Train Epoch: 9 [6144/14034 (44%)]       Loss: 0.482077
Train Epoch: 9 [7168/14034 (51%)]       Loss: 1.188591
Train Epoch: 9 [8192/14034 (58%)]       Loss: 0.302687
Train Epoch: 9 [9216/14034 (66%)]       Loss: 1.234665
Train Epoch: 9 [10240/14034 (73%)]      Loss: 0.648178
Train Epoch: 9 [11264/14034 (80%)]      Loss: 1.265709
Train Epoch: 9 [12288/14034 (87%)]      Loss: 0.429868
Train Epoch: 9 [13312/14034 (95%)]      Loss: 1.973314

Test set: Average loss: 0.0512, Accuracy: 2407/3000 (80%)

Train Epoch: 10 [0/14034 (0%)]  Loss: 0.750757
Train Epoch: 10 [1024/14034 (7%)]       Loss: 0.614832
Train Epoch: 10 [2048/14034 (15%)]      Loss: 1.201490
Train Epoch: 10 [3072/14034 (22%)]      Loss: 1.083280
Train Epoch: 10 [4096/14034 (29%)]      Loss: 0.407021
Train Epoch: 10 [5120/14034 (36%)]      Loss: 1.076406
Train Epoch: 10 [6144/14034 (44%)]      Loss: 1.493588
Train Epoch: 10 [7168/14034 (51%)]      Loss: 0.347435
Train Epoch: 10 [8192/14034 (58%)]      Loss: 1.322266
Train Epoch: 10 [9216/14034 (66%)]      Loss: 1.545725
Train Epoch: 10 [10240/14034 (73%)]     Loss: 0.671002
Train Epoch: 10 [11264/14034 (80%)]     Loss: 0.762595
Train Epoch: 10 [12288/14034 (87%)]     Loss: 1.306381
Train Epoch: 10 [13312/14034 (95%)]     Loss: 0.908904

Test set: Average loss: 0.0406, Accuracy: 2468/3000 (82%)

Total time taken: 932 seconds
```

In [58]:

```python
model = models.resnet50(pretrained=True)
for param in model.parameters():
    param.requires_grad = False
num_features= model.fc.in_features
model.fc = nn.Linear(num_features, 6)


criterion = nn.CrossEntropyLoss().cuda()
#Define Adam Optimiser with a learning rate of 0.005
optimizer = optim.Adam(model.fc.parameters(), lr=0.005)

start = timeit.default_timer()
```

```
for epoch in range(1, 11):
    train(model, device, train_loader, criterion, optimizer, epoch)
    test(model, device, test_loader, criterion)
stop = timeit.default_timer()
print('Total time taken: {} seconds'.format(int(stop - start)) )
```

```
Train Epoch: 1 [0/14034 (0%)]	Loss: 1.717109
Train Epoch: 1 [1024/14034 (7%)]	Loss: 0.777127
Train Epoch: 1 [2048/14034 (15%)]	Loss: 0.626964
Train Epoch: 1 [3072/14034 (22%)]	Loss: 0.624273
Train Epoch: 1 [4096/14034 (29%)]	Loss: 0.360270
Train Epoch: 1 [5120/14034 (36%)]	Loss: 0.379281
Train Epoch: 1 [6144/14034 (44%)]	Loss: 0.882145
Train Epoch: 1 [7168/14034 (51%)]	Loss: 0.588367
Train Epoch: 1 [8192/14034 (58%)]	Loss: 0.573616
Train Epoch: 1 [9216/14034 (66%)]	Loss: 0.585919
Train Epoch: 1 [10240/14034 (73%)]	Loss: 0.716255
Train Epoch: 1 [11264/14034 (80%)]	Loss: 1.128740
Train Epoch: 1 [12288/14034 (87%)]	Loss: 0.419357
Train Epoch: 1 [13312/14034 (95%)]	Loss: 0.287916

Test set: Average loss: 0.0276, Accuracy: 2475/3000 (82%)

Train Epoch: 2 [0/14034 (0%)]	Loss: 0.294059
Train Epoch: 2 [1024/14034 (7%)]	Loss: 0.455158
Train Epoch: 2 [2048/14034 (15%)]	Loss: 0.454901
Train Epoch: 2 [3072/14034 (22%)]	Loss: 0.942425
Train Epoch: 2 [4096/14034 (29%)]	Loss: 0.556668
Train Epoch: 2 [5120/14034 (36%)]	Loss: 0.790623
Train Epoch: 2 [6144/14034 (44%)]	Loss: 0.671098
Train Epoch: 2 [7168/14034 (51%)]	Loss: 0.395310
Train Epoch: 2 [8192/14034 (58%)]	Loss: 0.320078
Train Epoch: 2 [9216/14034 (66%)]	Loss: 0.726991
Train Epoch: 2 [10240/14034 (73%)]	Loss: 0.363501
Train Epoch: 2 [11264/14034 (80%)]	Loss: 0.448843
Train Epoch: 2 [12288/14034 (87%)]	Loss: 0.782134
Train Epoch: 2 [13312/14034 (95%)]	Loss: 0.834585

Test set: Average loss: 0.0366, Accuracy: 2371/3000 (79%)

Train Epoch: 3 [0/14034 (0%)]	Loss: 0.425066
Train Epoch: 3 [1024/14034 (7%)]	Loss: 0.862759
Train Epoch: 3 [2048/14034 (15%)]	Loss: 0.291625
Train Epoch: 3 [3072/14034 (22%)]	Loss: 1.163757
Train Epoch: 3 [4096/14034 (29%)]	Loss: 0.469650
Train Epoch: 3 [5120/14034 (36%)]	Loss: 1.060974
Train Epoch: 3 [6144/14034 (44%)]	Loss: 0.095958
Train Epoch: 3 [7168/14034 (51%)]	Loss: 0.607232
Train Epoch: 3 [8192/14034 (58%)]	Loss: 0.720122
Train Epoch: 3 [9216/14034 (66%)]	Loss: 0.643293
Train Epoch: 3 [10240/14034 (73%)]	Loss: 0.551962
Train Epoch: 3 [11264/14034 (80%)]	Loss: 0.719411
Train Epoch: 3 [12288/14034 (87%)]	Loss: 1.083867
Train Epoch: 3 [13312/14034 (95%)]	Loss: 0.679022

Test set: Average loss: 0.0340, Accuracy: 2414/3000 (80%)

Train Epoch: 4 [0/14034 (0%)]	Loss: 0.696792
Train Epoch: 4 [1024/14034 (7%)]	Loss: 0.917500
Train Epoch: 4 [2048/14034 (15%)]	Loss: 0.167900
```

```
Train Epoch: 4 [3072/14034 (22%)]       Loss: 0.602923
Train Epoch: 4 [4096/14034 (29%)]       Loss: 1.203177
Train Epoch: 4 [5120/14034 (36%)]       Loss: 0.290445
Train Epoch: 4 [6144/14034 (44%)]       Loss: 0.507768
Train Epoch: 4 [7168/14034 (51%)]       Loss: 1.012507
Train Epoch: 4 [8192/14034 (58%)]       Loss: 0.880904
Train Epoch: 4 [9216/14034 (66%)]       Loss: 0.187370
Train Epoch: 4 [10240/14034 (73%)]      Loss: 0.405398
Train Epoch: 4 [11264/14034 (80%)]      Loss: 0.685888
Train Epoch: 4 [12288/14034 (87%)]      Loss: 0.345292
Train Epoch: 4 [13312/14034 (95%)]      Loss: 0.891951

Test set: Average loss: 0.0265, Accuracy: 2512/3000 (84%)

Train Epoch: 5 [0/14034 (0%)]   Loss: 0.357024
Train Epoch: 5 [1024/14034 (7%)]        Loss: 0.685541
Train Epoch: 5 [2048/14034 (15%)]       Loss: 0.571120
Train Epoch: 5 [3072/14034 (22%)]       Loss: 0.637953
Train Epoch: 5 [4096/14034 (29%)]       Loss: 0.426983
Train Epoch: 5 [5120/14034 (36%)]       Loss: 0.589666
Train Epoch: 5 [6144/14034 (44%)]       Loss: 0.355986
Train Epoch: 5 [7168/14034 (51%)]       Loss: 0.750774
Train Epoch: 5 [8192/14034 (58%)]       Loss: 0.733743
Train Epoch: 5 [9216/14034 (66%)]       Loss: 0.154333
Train Epoch: 5 [10240/14034 (73%)]      Loss: 0.544956
Train Epoch: 5 [11264/14034 (80%)]      Loss: 0.345254
Train Epoch: 5 [12288/14034 (87%)]      Loss: 0.562934
Train Epoch: 5 [13312/14034 (95%)]      Loss: 0.508047

Test set: Average loss: 0.0249, Accuracy: 2510/3000 (84%)

Train Epoch: 6 [0/14034 (0%)]   Loss: 0.574429
Train Epoch: 6 [1024/14034 (7%)]        Loss: 1.071842
Train Epoch: 6 [2048/14034 (15%)]       Loss: 0.308824
Train Epoch: 6 [3072/14034 (22%)]       Loss: 0.481676
Train Epoch: 6 [4096/14034 (29%)]       Loss: 0.684890
Train Epoch: 6 [5120/14034 (36%)]       Loss: 1.029744
Train Epoch: 6 [6144/14034 (44%)]       Loss: 0.781261
Train Epoch: 6 [7168/14034 (51%)]       Loss: 0.044908
Train Epoch: 6 [8192/14034 (58%)]       Loss: 0.690256
Train Epoch: 6 [9216/14034 (66%)]       Loss: 0.686488
Train Epoch: 6 [10240/14034 (73%)]      Loss: 1.326398
Train Epoch: 6 [11264/14034 (80%)]      Loss: 0.180769
Train Epoch: 6 [12288/14034 (87%)]      Loss: 0.311545
Train Epoch: 6 [13312/14034 (95%)]      Loss: 0.345199

Test set: Average loss: 0.0229, Accuracy: 2580/3000 (86%)

Train Epoch: 7 [0/14034 (0%)]   Loss: 0.537391
Train Epoch: 7 [1024/14034 (7%)]        Loss: 0.258208
Train Epoch: 7 [2048/14034 (15%)]       Loss: 0.768453
Train Epoch: 7 [3072/14034 (22%)]       Loss: 0.403469
Train Epoch: 7 [4096/14034 (29%)]       Loss: 1.846524
Train Epoch: 7 [5120/14034 (36%)]       Loss: 0.929776
Train Epoch: 7 [6144/14034 (44%)]       Loss: 0.625378
Train Epoch: 7 [7168/14034 (51%)]       Loss: 1.933435
Train Epoch: 7 [8192/14034 (58%)]       Loss: 0.831787
Train Epoch: 7 [9216/14034 (66%)]       Loss: 1.013918
Train Epoch: 7 [10240/14034 (73%)]      Loss: 0.859825
Train Epoch: 7 [11264/14034 (80%)]      Loss: 0.814881
```

```
Train Epoch: 7 [12288/14034 (87%)]      Loss: 1.321859
Train Epoch: 7 [13312/14034 (95%)]      Loss: 0.477926

Test set: Average loss: 0.0296, Accuracy: 2469/3000 (82%)

Train Epoch: 8 [0/14034 (0%)]   Loss: 0.563002
Train Epoch: 8 [1024/14034 (7%)]        Loss: 0.768713
Train Epoch: 8 [2048/14034 (15%)]       Loss: 0.283069
Train Epoch: 8 [3072/14034 (22%)]       Loss: 0.803400
Train Epoch: 8 [4096/14034 (29%)]       Loss: 0.624199
Train Epoch: 8 [5120/14034 (36%)]       Loss: 0.525553
Train Epoch: 8 [6144/14034 (44%)]       Loss: 0.640880
Train Epoch: 8 [7168/14034 (51%)]       Loss: 0.665316
Train Epoch: 8 [8192/14034 (58%)]       Loss: 0.635447
Train Epoch: 8 [9216/14034 (66%)]       Loss: 1.046673
Train Epoch: 8 [10240/14034 (73%)]      Loss: 1.242952
Train Epoch: 8 [11264/14034 (80%)]      Loss: 0.239960
Train Epoch: 8 [12288/14034 (87%)]      Loss: 0.411483
Train Epoch: 8 [13312/14034 (95%)]      Loss: 1.484390

Test set: Average loss: 0.0287, Accuracy: 2488/3000 (83%)

Train Epoch: 9 [0/14034 (0%)]   Loss: 0.496528
Train Epoch: 9 [1024/14034 (7%)]        Loss: 0.145967
Train Epoch: 9 [2048/14034 (15%)]       Loss: 0.591382
Train Epoch: 9 [3072/14034 (22%)]       Loss: 0.779742
Train Epoch: 9 [4096/14034 (29%)]       Loss: 0.488741
Train Epoch: 9 [5120/14034 (36%)]       Loss: 0.419754
Train Epoch: 9 [6144/14034 (44%)]       Loss: 0.237933
Train Epoch: 9 [7168/14034 (51%)]       Loss: 0.789222
Train Epoch: 9 [8192/14034 (58%)]       Loss: 0.409392
Train Epoch: 9 [9216/14034 (66%)]       Loss: 0.910044
Train Epoch: 9 [10240/14034 (73%)]      Loss: 1.033690
Train Epoch: 9 [11264/14034 (80%)]      Loss: 0.712912
Train Epoch: 9 [12288/14034 (87%)]      Loss: 0.660020
Train Epoch: 9 [13312/14034 (95%)]      Loss: 0.485562

Test set: Average loss: 0.0289, Accuracy: 2495/3000 (83%)

Train Epoch: 10 [0/14034 (0%)]  Loss: 0.750603
Train Epoch: 10 [1024/14034 (7%)]       Loss: 0.775502
Train Epoch: 10 [2048/14034 (15%)]      Loss: 0.553951
Train Epoch: 10 [3072/14034 (22%)]      Loss: 1.051222
Train Epoch: 10 [4096/14034 (29%)]      Loss: 0.327797
Train Epoch: 10 [5120/14034 (36%)]      Loss: 0.331540
Train Epoch: 10 [6144/14034 (44%)]      Loss: 0.560924
Train Epoch: 10 [7168/14034 (51%)]      Loss: 1.464945
Train Epoch: 10 [8192/14034 (58%)]      Loss: 0.646801
Train Epoch: 10 [9216/14034 (66%)]      Loss: 0.280804
Train Epoch: 10 [10240/14034 (73%)]     Loss: 0.841801
Train Epoch: 10 [11264/14034 (80%)]     Loss: 0.424293
Train Epoch: 10 [12288/14034 (87%)]     Loss: 0.940271
Train Epoch: 10 [13312/14034 (95%)]     Loss: 0.706470

Test set: Average loss: 0.0271, Accuracy: 2521/3000 (84%)

Total time taken: 957 seconds
```

In [ ]:
```
model = timm.create_model('inception_resnet_v2', pretrained=True, num_classes=6)
```

In [40]:
```
model
```

Out[40]:
```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=Fa
lse)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=Fa
lse)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=Fa
lse)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
      (relu): ReLU(inplace=True)
    )
  )
  (layer2): Sequential(
    (0): Bottleneck(
```

```
        (conv1): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=
False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
        (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
        (relu): ReLU(inplace=True)
        (downsample): Sequential(
          (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
        )
      )
      (1): Bottleneck(
        (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=
False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
        (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
        (relu): ReLU(inplace=True)
      )
      (2): Bottleneck(
        (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=
False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
        (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
        (relu): ReLU(inplace=True)
      )
      (3): Bottleneck(
        (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=
False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
        (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
        (relu): ReLU(inplace=True)
      )
    )
    (layer3): Sequential(
      (0): Bottleneck(
        (conv1): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
```

```
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=
False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats
=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(512, 1024, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats
=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=
False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats
=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=
False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats
=True)
      (relu): ReLU(inplace=True)
    )
    (3): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=
False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats
=True)
      (relu): ReLU(inplace=True)
    )
    (4): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=
```

```
False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats
=True)
      (relu): ReLU(inplace=True)
    )
    (5): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=
False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats
=True)
      (relu): ReLU(inplace=True)
    )
  )
  (layer4): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=
False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats
=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats
=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=
False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats
=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=
False)
```

```
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats
=True)
      (relu): ReLU(inplace=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=2048, out_features=6, bias=True)
)
```

Validate the Model

In [59]:

```
valid_transforms = transforms.Compose([transforms.Resize(224),
                                       transforms.ToTensor(),
                                      ])
```

In [115…

```
def predict_image(image):
    image_tensor = valid_transforms(image).float()
    image_tensor = image_tensor.unsqueeze_(0)
    input = Variable(image_tensor)
    input = input.to(device)
    output = model(input)
    index = output.data.cpu().numpy().argmax()
    vector={0:'buildings',1:'forest',2:'glacier',3:'mountain',4:'sea',5:'street'}
    obj=vector[index]


    return obj
```

In [116…

```
lis= [
    "../input/intel-image-classification/seg_pred/seg_pred/10004.jpg",
    "../input/intel-image-classification/seg_pred/seg_pred/10043.jpg",
    "../input/intel-image-classification/seg_pred/seg_pred/10045.jpg",
    "../input/intel-image-classification/seg_pred/seg_pred/10054.jpg",
    "../input/intel-image-classification/seg_pred/seg_pred/1008.jpg",
    "../input/intel-image-classification/seg_pred/seg_pred/10059.jpg",
    "../input/intel-image-classification/seg_pred/seg_pred/10069.jpg",
]
```

In [86]:

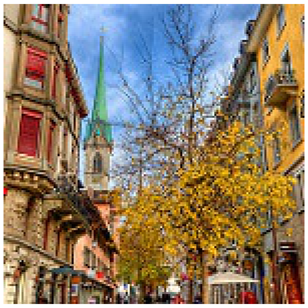
```
for image in lis:
    im = Image.open(image)
    print(predict_image(im))
    im
```

```
street
sea
street
glacier
street
forest
sea
```

In [130…

```
im0 = Image.open(lis[0])
im0
```

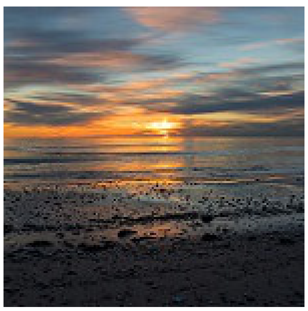
Out[130…



In [131…

```
im1 = Image.open(lis[1])
im1
```

Out[131…



In [132…

```
im1 = Image.open(lis[2])
im1
```

Out[132…

In [133…

```
im1 = Image.open(lis[3])
im1
```

Out[133…

In [134…

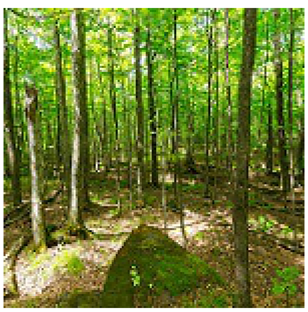
```
im1 = Image.open(lis[4])
im1
```

Out[134…

In [135…

```
im1 = Image.open(lis[5])
im1
```

Out[135…



In [136…

```
im1 = Image.open(lis[6])
im1
```

Out[136…

In [137…

```
pred_path='../input/intel-image-classification/seg_pred'
valid_path=os.path.join(pred_path,'seg_pred')
```

In [ ]:

```

```